

Restricted Recurrent Neural Tensor Networks: Exploiting Word Frequency and Compositionality for Increased Model Capacity and Performance With No Computational Overhead

Alexandre Salle Aline Villavicencio

Institute of Informatics

Universidade Federal do Rio Grande do Sul

Porto Alegre, Brazil

{atsalle, avillavicencio}@inf.ufrgs.br

Abstract

Increasing the capacity of recurrent neural networks (RNN) usually involves augmenting the size of the hidden layer, resulting in a significant increase of computational cost. An alternative is the recurrent neural tensor network (RNTN), which increases capacity by employing distinct hidden layer weights for each vocabulary word. However, memory usage scales linearly with vocabulary size, which can reach millions for word-level language models. In this paper, we introduce restricted recurrent neural tensor networks (r-RNTN) which reserve distinct hidden layer weights for frequent vocabulary words while sharing a single set of weights for infrequent words. Perplexity evaluations show that r-RNTNs improve language model performance over standard RNNs using only a small fraction of the parameters of unrestricted RNTNs.

1 Introduction

Sequence modeling is an important part of natural language processing, with applications in machine translation, speech recognition, and language modeling. Recurrent neural networks (RNN), which compute their next output conditioned on a previously stored hidden state, are a natural solution to sequence modeling.

Mikolov et al. (2010) applied RNNs to word-level language modeling (we refer to this model as s-RNN), outperforming traditional n-gram methods. One issue with the s-RNN is increasing its capacity (number of tunable parameters) to be able to model more complex distributions, as increasing the size H of the hidden (or recurrent) layer

results in a significant increase in computational cost, which is $O(H^2)$.

Sutskever et al. (2011) increased the performance of their character-level language model by proposing a multiplicative RNN (m-RNN), the factored approximation of a recurrent neural tensor network (RNTN), which maps each symbol to separate hidden layer weights (referred to as recurrence matrices from hereon). Besides increasing model capacity while keeping computation constant, this approach has another motivation: viewing the RNN’s hidden state as being transformed by each new symbol in the sequence, it is intuitive that different symbols will transform the network’s hidden state in different ways (Sutskever et al., 2011). Various studies on compositionality similarly argue that some words are better modeled by matrices than by vectors (Baroni and Zamparelli, 2010; Socher et al., 2012). Unfortunately, having separate recurrence matrices for each symbol requires memory that is linear in the symbol vocabulary size ($|V|$). This is not an issue for character-level models, which have small vocabularies, but is prohibitive for word-level models which can have vocabulary size in the millions if we consider surface forms.

In this paper, we propose the Restricted RNTN (r-RNTN) which uses only $K < |V|$ recurrence matrices. Given that $|V|$ words must be assigned K matrices, we map the most frequent $K - 1$ words to the first $K - 1$ matrices, and share the K -th matrix among the remaining words. This mapping is driven by the statistical intuition that frequent words are more likely to appear in diverse contexts and so require richer modeling, and by the greater presence of predicates and function words among the most frequent words in standard corpora like COCA (Davies, 2009). As a result, adding K matrices to the s-RNN both increases model capacity and satisfies the idea that

in modeling compositionality some words are better represented by matrices. Results show that r-RNTNs improve language model performance over s-RNNs even for small K with no computational overhead, and even for small K approximate the performance of RNTNs using a fraction of the parameters. This paper is organized as follows: we review related work (§2) and then present r-RNTNs (§3), describing the evaluation method (§4) and discussing results (§5), concluding with final remarks and suggestions for future work.

2 Related Work

We focus on related work that addresses language modeling via RNNs, word representation, and conditional computation.

Given a sequence of words (x_1, \dots, x_T) , a language model gives the probability $P(x_t|x_{1..t-1})$ for $t \in [1, T]$. Using a RNN, Mikolov et al. (2010) created the s-RNN language model given by:

$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \quad (1)$$

$$P(x_t|x_{1..t-1}) = x_t^T \text{Softmax}(W_{oh}h_t + b_o) \quad (2)$$

where h_t is the hidden state, $\sigma(z)$ is the logistic function, W_{hx} is the $H \times V$ embedding matrix, W_{hh} is the $H \times H$ recurrence matrix, and b_h and b_o are bias terms. Computation is $O(H^2)$, so increasing model capacity by increasing H quickly becomes intractable.

The RNTN proposed by Sutskever et al. (2011) is nearly identical to the s-RNN, but the recurrence matrix in eq. (1) is replaced by a tensor as follows:

$$h_t = \sigma(W_{hx}x_t + W_{hh}^{i(x_t)}h_{t-1} + b_h) \quad (3)$$

where $i(z)$ maps a hot-one encoded vector to its integer representation. The W_{hh} tensor is therefore composed of $|V|$ recurrence matrices, and at each step of sequence processing the matrix corresponding to the current input is used to transform the hidden state. The authors also proposed m-RNN, a factorization of the $W_{hh}^{i(x_t)}$ tensor into $W_{lh}\text{diag}(v_{x_t})W_{rh}$ to reduce the number of parameters, where v_{x_t} is a *factor* vector of the current input x_t , but like the RNTN, memory still grows linearly with $|V|$. The RNTN has the property that input symbols have both a vector representation given by the embedding and a matrix representation given by the recurrence matrix, unlike the s-RNN where symbols are limited to vector representations.

The integration of both vector and matrix representations has been discussed but with a focus on representation learning and not sequence modeling (Baroni and Zamparelli, 2010; Socher et al., 2012). For instance, Baroni and Zamparelli (2010) argue for nouns to be represented as vectors and adjectives as matrices, where for a given attributive adjective its representation is derived from its occurrences with different nouns in corpora (e.g. *green apple/wall/leaf*).

Irsoy and Cardie (2014) used m-RNNs for the task of sentiment classification and obtained equal or better performance than s-RNNs. Methods that use conditional computation (Cho and Bengio, 2014; Bengio et al., 2015; Shazeer et al., 2017) are similar to RNTNs and r-RNTNs, but rather than use a static mapping, these methods train gating functions which do the mapping. Although these methods can potentially learn better policies, they are significantly more complex and offer less insight than our method into why a given mapping is better than some other.

Whereas our work is concerned with updating the network’s hidden state, Chen et al. (2015) introduce a technique that better approximates the output layer’s Softmax function by allocating more parameters to frequent words.

There are many other improvements to s-RNNs, the most notable of which is the use of Long Term Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997), but these are too numerous to list here and for the most part complementary to our proposed method. In section 6 we discuss how r-RNTNs might be applied to LSTMs.

3 Restricted Recurrent Neural Tensor Networks

To balance expressiveness and computational cost, we propose restricting the size of the recurrence tensor in the RNTN such that memory does not grow linearly with vocabulary size while still keeping dedicated matrix representations for a subset of words in the vocabulary. We call these Restricted Recurrent Neural Tensor Networks (r-RNTN), which modify eq. (3) as follows:

$$h_t = \sigma(W_{hx}x_t + W_{hh}^{f(i(x_t))}h_{t-1} + b_h^{f(i(x_t))}) \quad (4)$$

where W_{hh} is a tensor of $K < |V|$ matrices of size $H \times H$, b_h is a $K \times H$ bias matrix with rows

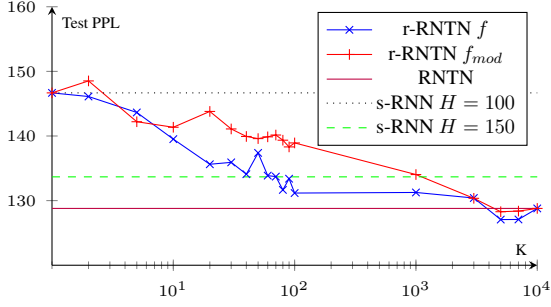


Figure 1: PTB test set perplexity as K varies from 1 to 10000. At $K = 100$, the r-RNTN with f mapping already closely approximates the much bigger RNTN, with little gain for bigger K , showing that dedicated matrices should be reserved for frequent words as hypothesized.

indexed by f . The function $f(w)$ maps each vocabulary word to an integer between 1 and K .

We use the following definition for f :

$$f(w) = \min(\text{rank}(w), K) \quad (5)$$

where $\text{rank}(w)$ is the rank of word w when the vocabulary is sorted by decreasing order of unigram frequency.

This is an intuitive choice because words which appear more often in the corpus tend to have more variable contexts, so it makes sense to dedicate a large part of model capacity to them. A second argument is that frequent words tend to be predicates and function words. We can imagine that predicates and function words transform the meaning of the current hidden state of the RNN through matrix multiplication, whereas nouns, for example, add meaning through vector addition, following Baroni and Zamparelli (2010).

4 Materials

As done in other language modeling work (Mikolov et al., 2011; Mnih and Teh, 2012; Zaremba et al., 2014; Mikolov et al., 2014; Shazeer et al., 2017), we evaluate s-RNNs, RNTNs, and r-RNTNs by training and measuring model perplexity (PPL) on the Penn Treebank (PTB) corpus (Marcus et al., 1994) using the same pre-processing as Mikolov et al. (2011). This corpus is split into training, validation, and test sets containing 929K words, 73K words, and 82K words respectively. The vocabulary is restricted to the most frequent 10000 words and all other words are mapped to the “<unk>” token.

For an r-RNTN with $H = 100$, we vary the tensor size K from 1, which corresponds to the s-RNN, all the way up to 10000, which corresponds to the unrestricted RNTN. As a simple way to evaluate our choice of rank-based mapping function f , we compare it to a pseudo-random variant:

$$f_{mod}(w) = \text{rank}(w) \bmod K \quad (6)$$

We also compare results to 1) an s-RNN with $H = 150$, which has the same number of parameters as an r-RNTN with $H = 100$ and $K = 100$. 2) an m-RNN with $H = 100$ with the size of factor vectors set to 100 to match this same number of parameters. An additional r-RNTN with $H = 150$ is trained to show that performance scales with H as well.

We split each sentence into 20 word subsequences and run stochastic gradient descent via backpropagation through time (Werbos, 1990) for 20 steps, only resetting the RNN’s hidden state between sentences. We did not use mini-batching so as to simplify the implementation. The learning rate was initialized at 0.1 and halved when the ratio of the validation perplexity between successive epochs was less than 1.003. We early-stopped training when validation improvement fell below this ratio for 5 consecutive epochs.

We found that both the r-RNTN and RNTN were easily able to overfit the training data. While $L2$ regularization did little to remedy this issue, we were able to overcome it by using Dropout (Srivastava et al., 2014) with $p = 0.5$ after the activations h_t of the hidden layer.

To validate our proposed method, we also evaluated r-RNTNs using the much larger text8¹ corpus, which consists of the first 100MB from Wikipedia. We used the first 95MB as the training set, the next 5MB as the validation set, and the remaining 5MB as the test set. Vocabulary was restricted to words which appeared at least 10 times in the training set, for a total of 37751 words. On this corpus, we set $K = 376$ so that the r-RNTN has the same number of parameters as the s-RNN with $H = 150$.

5 Results

Table 1 shows results on both PTB and text8 as H varies. The results on PTB as K is varied are presented in fig. 1. Comparing the r-RNTN to the baseline s-RNN with $H = 100$ in fig. 1, we see

¹<http://mattmahoney.net/dc/textdata.html>

Method	H	PTB		text8	
		# Params	Test PPL	# Params	Test PPL
s-RNN	100	2M	146.7	7.6M	236.4
r-RNTN f	100	3M	131.2	11.4M	190.1
RNTN	100	103M	128.8	388M	-
m-RNN	100	3M	164.2	11.4M	895.0
s-RNN	150	3M	133.7	11.4M	207.9
r-RNTN f	150	5.3M	126.4	19.8M	171.7

Table 1: Comparison of validation and test set perplexity for r-RNTNs with f mapping ($K = 100$ for PTB, $K = 376$ for text8) versus s-RNNs and m-RNN. r-RNTNs with the same H as corresponding s-RNNs significantly increase model capacity and performance with no computational cost. The RNTN was not run on text8 due to the number of parameters required.

that as model capacity grows with K , test set perplexity drops, showing that r-RNTN is an effective way to increase model capacity with no additional computational cost. As expected, the f mapping outperforms the pseudo-random baseline f_{mod} mapping at smaller K . As K increases, we see a convergence of both mappings. This can be explained by the fact that at large K , the most frequent words are sharing matrices with infrequent words because of the modulus operation in eq. (6). As infrequent words are rarely observed, frequent words dominate the matrix updates and approximate having distinct matrices, as they would have with the f mapping.

It is remarkable that even with K as small as 100, the r-RNTN approaches the performance of the RNTN with a small fraction of the parameters. This reinforces our hypothesis that complex transformation modeling afforded by distinct matrices is needed for frequent words, but not so much for infrequent words which can be well represented by a shared matrix and a distinct vector embedding. As shown in table 1, with an equal number of parameters, the r-RNTN with f mapping outperforms the s-RNN with a bigger hidden layer. It appears that heuristically allocating increased model capacity as done by the f based r-RNTN is a better way to increase performance than simply increasing hidden layer size, which also incurs a computational penalty.

Although m-RNNs have been successfully employed in character-level language models with small vocabularies, they are seldom used in word-level models. The poor results shown in table 1 could explain why.²

²It should be noted that Sutskever et al. (2011) suggest m-RNNs would be better optimized using second-order gradient

Our results show significant improvements to s-RNNs, confirming the advantages of distinct representations. Even if they fall short of state of the art methods which utilize LSTMs (Jozefowicz et al., 2016), r-RNTNs and LSTMs are not mutually exclusive, and in the next section we discuss how we plan to integrate both.

6 Conclusion and Future Work

In this paper, we proposed restricted recurrent neural tensor networks, a model that restricts the size of recurrent neural tensor networks by mapping frequent words to distinct matrices and infrequent words to shared matrices. r-RNTNs were motivated by the need to increase RNN model capacity without increasing computational costs, while also satisfying the compositionality ideas that some words are better modeled by matrices rather than vectors (Baroni and Zamparelli, 2010; Socher et al., 2012). We achieved both goals by pruning the size of the recurrent neural tensor network described by Sutskever et al. (2011) via sensible word-to-matrix mapping. Results validated our hypothesis that frequent words benefit from richer, dedicated modeling as reflected in large perplexity improvements for low values of K .

r-RNTNs are mostly complementary to other improvements in RNNs. In future work, we will use r-RNTNs with LSTMs (which are employed by state of the art methods) to confirm that similar gains can be obtained. Whereas s-RNNs have a single recurrence matrix and directly use a word’s embedding, LSTMs have four recurrence matrices and four input matrices. Our work will consist of determining which of these matrices, when made

descent methods, whereas we employed only first-order gradients in all models we trained to make a fair comparison.

input-specific, increase model performance.

References

- Marco Baroni and Roberto Zamparelli. 2010. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1183–1193.
- Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. 2015. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*.
- Welin Chen, David Grangier, and Michael Auli. 2015. Strategies for training large vocabulary neural language models. *arXiv preprint arXiv:1512.04906*.
- Kyunghyun Cho and Yoshua Bengio. 2014. Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning. *arXiv preprint arXiv:1406.7362*.
- Mark Davies. 2009. The 385+ million word corpus of contemporary american english (1990–2008+): Design, architecture, and linguistic insights. *International journal of corpus linguistics* 14(2):159–190.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.
- Ozan Irsoy and Claire Cardie. 2014. Modeling compositionality with multiplicative recurrent neural networks. *arXiv preprint arXiv:1412.6577*.
- Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.
- Mitchell P. Marcus, Beatrice Santorini, and Mary A. Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19(2):313–330.
- T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Cernocký. 2011. Empirical evaluation and combination of advanced language modeling techniques. In *INTERSPEECH*. pages 605–608.
- Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc’Aurelio Ranzato. 2014. Learning longer memory in recurrent neural networks. *arXiv preprint arXiv:1412.7753*.
- Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*. pages 1045–1048.
- Andriy Mnih and Yee W. Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*.
- Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarczyk, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.
- Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, pages 1201–1211.
- Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.
- Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. pages 1017–1024.
- Paul J. Werbos. 1990. [Backpropagation through time: what it does and how to do it](https://doi.org/10.1109/5.58337). *Proceedings of the IEEE* 78(10):1550–1560. <https://doi.org/10.1109/5.58337>.
- Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.